Strategic Pattern Discovery in RTS-games for E-Sport with Sequential Pattern Mining

Guillaume Bosc[1], Mehdi Kaytoue[1], Chedy Raïssi[2], and Jean-François Boulicaut[1]

[1]Université de Lyon, CNRS, INSA-Lyon, LIRIS, UMR5205, F-69621, France
[2]INRIA Nancy Grand Est, France
firstname.lastname@insa-lyon.fr, raissi@inria.fr

Abstract. Electronic sport, or e-sport, denotes the extreme practice of video games where so-called cyber-athletes compete in world-wide tournaments. As for any sport, such professionals are surrounded by sponsors and practice within professional teams. These professional games are even broadcast by commentators over specialized TV channels. StarCraft II (Blizzard Entertainment) is one of the most competitive video game and has now its own world-wide players ranking system (based on the well-known ELO system) and annual world cup competition series (WCS) with a US$1.6 millions prize pool for the year 2013. Each match between two opponents can be recorded as an exhaustive and noisy sequence of actions. Analyzing these sequences yields an important outcome for game strategy prediction and in-depth game understanding. In this work we report a preliminary study on StarCraft II professional players strategies' discovery based on sequential pattern mining.

1 Introduction

The progressing digitization of modern societies is witnessed in many domains and has irrevocably impacted all aspects of human behaviors. In the wake of these radical digital and social evolutions, the entertainment industry underwent quick mutations to benefit from the widely spread usage of electronic devices (computers, play stations, smart-phones, etc.). Over the last decade, the video games industry became one of the most lucrative business ever developed with many examples such as Ubisoft budgeting 40 millions euros to develop "Assassin's creed III" or "Angry birds" (Rovio Entertainment) accounting more than 600 millions players. The competition spirit appeared in the early beginnings of video game developments [9]. Naturally, some people easily acquire (gaming) skills among which agility (mouse, keyboard interaction), creativity, reactivity and strategical thinking. To push those skills to the highest, they perform an intense daily practice. These individuals compete in international tournaments, highlighting impressive skills that non daily practitioners cannot achieve. This new notion of competition over electronic games and its associated community is coined by the terms *"electronic-sport"* or *"e-sport"* and started in South Korea 15 years ago before spreading to Europe and North America over the last three years [9]. E-sport in general is composed of professionals, amateurs, teams, championships, commentators and sponsors. The main difference with classical sport is the usage of an electronic device as a support to compete[1]. This new sport paradigm is poised to challenge classical sports in term of audience in the next decade [5][2]. E-sport should attract many industrial actors and researchers in the next few years around video game analytics, or e-sport analytics, just like with traditional sports (as discussed during 2012 MIT Sloan Sport Conference, *eSports: The Future Of Competition*). In this context of digital competition, both sport analytic and artificial intelligence should meet to answer several problems. Indeed, as digital, e-sports easily leave (interaction) traces in great numbers on the Web (e.g. records of the games, statistics of the matches, spectators opinions and audience on social TV [5]). Match records, also called *replays*, contain all actions generated by the players. Strategies are hidden in such replays and eliciting them automatically can help for modeling players behaviors, predicting strategies, improving online match-making systems, commercial targeting, etc [2, 4, 8, 3, 12, 10, 11, 7]. In this article, we propose an approach for automatically discovering strategic patterns for one of the most competitive real-time strategy game (RTS), StarCraft II (Blizzard Entertainment, 2010). We investigate how sequential pattern mining [6, 13] can help strategy discovery from replays. In our case, a pattern represents a frequent series of game actions done by the two players extracted from a large database of games. We introduce a measure that describes the balance of a pattern (or a strategy), i.e. in which proportion it leads to victory. In pattern mining, this can be formalized with so-called emerging patterns when each object of the dataset is labeled either positively or negatively [1]. We deal with a hard problem, not studied in the literature, to the best of our knowledge, where the class information (either positive, or negative, for a win or a loss) needs to be encoded within the object description to discriminate both players, in contrast with a single class value for each sequence in the classical settings. We also show with an empirical evaluation that the patterns (and their balance measure) we found in about 100,000 replays can be helpful for game balance study, players modeling and winning prediction.

The rest of the paper is as follows. The principles of the StarCraft II game are presented in Section 2. Section 3 recalls notions on sequential pattern mining. We then present the problem of mining strategic patterns and our methodology in Section 4. We present experiments in Section 5 before concluding.

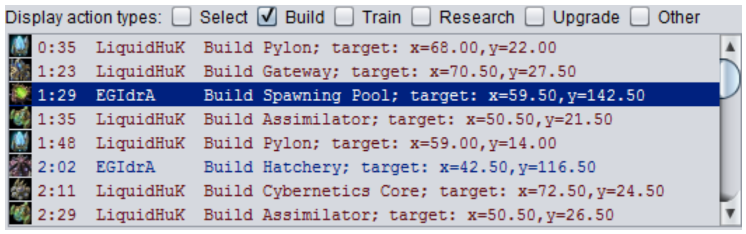
2 Principle of the RTS video game StarCraft II

A game of StarCraft II involves two players. Each player chooses a faction among the *"Zergs"* (Z), *"Protoss"* (P) and *"Terrans"* (T): there are 6 different possible

[1] *"E-sport is one of the most popular trend in video games and is rapidly attracting the core 18-34 male demographic in greater numbers than any other medium or category"*, said Jim Lanzone, president of CBS Interactive.

[2] *"Major League Gaming (MLG)'s live audience is growing in popularity to rival some of the most popular traditional sporting events with a highly engaged fan base world-wide"*, said Sundance DiGiovanni, CEO and Co-Founder of MLG.

combinations of matches. Each combination involves different strategies. During a game, two players are battling on a map (aerial view), controlling buildings and units to gather resources, train an army with the final goal of winning by annihilating the opponent's forces[3]. Such actions (training, building, moving, attacking) are done in real-time, see Figure 1 for an example. Each faction (Z,P,T) allows different units and buildings with distinctive weaknesses and strengths following a *"rock-paper-scissors"* principle. A strategy is hidden in large sequences of actions generated by players called *replays*. Actually, the name "replay" comes from the fact that it materializes all actions allowing the game engine to replay the game anytime. A high proportion of the actions are noisy: a replay does not store the result of an action, i.e. if/how it changed the state of the game. For example, if a player orders the construction of a building while it does not have the resources to realize it, this action is still stored in the replay but the state of the game has not changed since construction is impossible. Moreover, players are constrained by the *"fog of war"* which reduces the player's field of vision of the game by only revealing terrain features and enemy units only if they pass near a player unit or building. Areas of the map which are out of the field of vision are subject to a shroud through which only terrain is visible, but not changes in enemy units or bases. Then, scouting the enemy's base is an action a player often needs to perform to adapt his strategy according to the opposing player's one. The fog of war is very similar to the *"imperfect information"* notion that appears in game theory or economics.


Display action types: Select [✓ Build] Train Research Upgrade Other

```
0:35  LiquidHuK  Build Pylon; target: x=68.00,y=22.00
1:23  LiquidHuK  Build Gateway; target: x=70.50,y=27.50
1:29  EGIdrA     Build Spawning Pool; target: x=59.50,y=142.50
1:35  LiquidHuK  Build Assimilator; target: x=50.50,y=21.50
1:48  LiquidHuK  Build Pylon; target: x=59.00,y=14.00
2:02  EGIdrA     Build Hatchery; target: x=42.50,y=116.50
2:11  LiquidHuK  Build Cybernetics Core; target: x=72.50,y=24.50
2:29  LiquidHuK  Build Assimilator; target: x=50.50,y=26.50
```

Fig. 1. An example of build-actions realized during a game.

3 Preliminaries on sequential pattern mining

We use the notations from [13]. Let $\mathcal{I}$ be a set of *items*. An *itemset* is a subset of $\mathcal{I}$. A *sequence* s is a non empty ordered list of *itemsets*: $s = \langle t_1 t_2 ... t_n \rangle$ where $t_j \subseteq \mathcal{I}$ for all $j \in \{1, ..., n\}$. A *sequence* $\alpha = \langle a_1 a_2 ... a_n \rangle$ is a *sub-sequence* of an other *sequence* $\beta = \langle b_1 b_2 ... b_m \rangle$, denoted as $\alpha \sqsubseteq \beta$, if and only if $\exists j_1, j_2, ..., j_n$ such as $1 \leq j_1 < j_2 < ... < j_n \leq m$ and $a_1 \subseteq b_{j_1}, a_2 \subseteq b_j, ..., a_n \subseteq b_{j_n}$. Given a *sequence database* $\mathcal{D} = \{s_1, s_2, ..., s_n\}$, the *support* of an arbitrary sequence α (over $\mathcal{I}$) is defined as the id of sequences in $\mathcal{D}$ which contain α: $support_{\mathcal{D}}(\alpha) = \{s \mid \alpha \sqsubseteq s,\ s \in \mathcal{D}\}$. A *sequence* α is said *frequent* in a database $\mathcal{D}$ if, given a *minimum support threshold minSupp*, $|support_{\mathcal{D}}(\alpha)| \geq minSupp$. Prefixspan is one of the reference algorithm for extracting frequent sequential patterns [13].

[3] http://en.wikipedia.org/wiki/Real-time_strategy

Example 1. The table on the right presents a sequence database where the set of items is $\mathcal{I} = \{a,\ b,\ c,\ d,\ e,\ f\}$. The sequence $\alpha = \langle abc \rangle$ is a sub-sequence of the sequences s_{10} and s_{40} but α is not a sub-sequence of sequences s_{20} and s_{30}. Moreover, the sequence $\beta = \langle \{df\}c \rangle$ is only a sub-sequence of the sequence s_{30}. With $minSupp = 3$, the sequence $\chi = \langle acc \rangle$ is a frequent sequential pattern because $support_{\mathcal{D}}(\chi) = \{s_{10}, s_{20}, s_{40}\}$. However the sequence $\delta = \langle a\{bc\}a \rangle$ is not frequent since $|support_{\mathcal{D}}(\delta)| = 2 < minSupp$.

id	description
s_{10}	$\langle a\{abc\}\{ac\}d\{cf\}\rangle$
s_{20}	$\langle \{ad\}c\{bc\}\{ae\}\rangle$
s_{30}	$\langle \{ef\}\{ab\}\{df\}cb\rangle$
s_{40}	$\langle eg\{af\}cbc\rangle$

4 Mining strategic patterns as sequential patterns

Given a corpus of StarCraft II replays, we consider the problem of extracting strategic patterns and evaluating (i.e., measure) how they lead or not to victory. To answer this problem, our methodology is decomposed as follows: (i) we encode a set of replays as a database of sequences, (ii) we mine frequent sequential patterns then (iii) we post-process these patterns to characterize and contrast their ability to discriminate winning from losing strategies.

We abstract the notion of replay in a way that is similar to *AI planning*: it represents two agents trying to succeed by performing inter-dependent actions in time. At the end of their interaction, one will fail, the other will succeed. The fact that actions of one agent are dependent of the other agent's actions (e.g. knowing the last action of the opponent can change one's strategy) leads us to derive one sequence with both agent actions instead of one sequence per agent. To be able to characterize if an agent is successful, we include success or failure in the sequence alphabet: an action a can appear in a sequence as leading to success, or to failure as final outcome. Hence, our sequence alphabet is composed of elements of the Cartesian product $Actions \times \{success, fail\}$. Furthermore, the window of time at which a typical action happens is very important in our application in competitive gaming. Hence, we define the alphabet $\mathcal{I}$ of our sequences as follow.

Definition 1 (Alphabet). *Consider a set of actions A available for both agents and a set of timestamps T, i.e. windows of time in which the actions occur. The alphabet of the sequences is defined as $\mathcal{I} = A \times T \times R$ where $R = \{success, fail\}$.*

Example 2. Thanks to this alphabet, we translate the replay sample given in Figure 1 as follows. We consider windows of 30 seconds and the player LiquidHuK is the winner of this game. We obtain $s = \langle \{(Pylon,\ 2,\ success)\}\ \{(Gateway,\ 3,\ success),\ (Spawning,\ 3,\ fail)\}\ \{(Assimilator,\ 4,\ success),\ (Pylon,\ 4,\ success)\}\ \{(Hatchery,\ 5,\ fail),\ (Cybernetics\ Core,\ 5,\ success),\ (Assimilator,\ 5,\ success)\}\rangle$ where the item $(Hatchery,\ 5,\ fail)$ means that the player who lost performed an action to build a Hatchery during the fifth window of time.

Consider now a database of sequences encoded from raw replays thanks to the alphabet presented above: a frequent sequential pattern gives us frequent series actions made by the agents, one of the agent will succeed at the end, the other will fail. In order to assess the interestingness of the resulting patterns, we define a measure that reflects the success or failure ability of the strategy materialized by the pattern. We define the notion of dual sequence first.

Definition 2 (Dual sequence). *The dual of a sequence s composed of items $(a,t,r) \in \mathcal{I}$ is the same sequence where any $(a,t,r) \in \mathcal{I}$ is replaced by $(a,t,R\backslash r)$.*

Consider a pattern $s = \langle (a, 3, fail), (b, 4, success) \rangle$ which means that an agent succeeds by doing the action b after the other agent did the action a. On the other hand, we have $\tilde{s} = \langle (a, 3, success), (b, 4, fail) \rangle$ which characterizes the opposite scenario: both agents do respectively the same actions, but the agent succeeding is not the same. The balance measure we introduce now represents the potential of strategy s to lead to success or failure.

Definition 3 (Balance measure). *Consider a frequent sequential pattern s and its dual $\tilde{s}$. We define the balance measure of s as:*

$$balance(s) = \frac{|support_{\mathcal{D}}(s)|}{|support_{\mathcal{D}}(s)| + |support_{\mathcal{D}}(\tilde{s})|}$$

The measure *balance* returns values in $[0; 1]$. When $balance(s) = 0.5$ the sequence s does not discriminate either success of fail: it is a **balanced** strategy. When $balance(s) = 1$ (resp. 0), the sequential pattern s is found only winning or losing. Note that $balance(s) + balance(\tilde{s}) = 1$. Note also that symmetric patterns, i.e. sequences where each itemset that contains the item $(a, t, r) \in \mathcal{I}$ also contains the dual item $(a, t, R\backslash r) \in \mathcal{I}$, have a balance of 0.5 by definition.

Algorithm. To compute the set of frequent patterns given a minimal support we use the PrefixSpan algorithm [6]. To compute the balance measure in post-processing, it is naively enough to compute $support_{\mathcal{D}}(\tilde{s})$ for each frequent pattern s by scanning the database $\mathcal{D}$. This approach is however not scalable as shown in the experiments. To allow a much faster computation of the *balance* measure, we proceed as follow. We build an auxiliary data structure when producing the sequences dataset from the rough replays, storing for each item $i \in \mathcal{I}$ the identifiers of the sequences that contain it, i.e. $\mathcal{D}\times\mathcal{I}$ in the worst case. Then, when building the tree-structure storing the patterns in PrefixSpan, we maintain for each pattern (node) the support of the dual sequence: at the first step by looking up in our data structure, in the other steps by intersecting the support of the dual item expanding the tree with the support of the previous dual sequence. As such, one can notice that there are redundant patterns: it happens that both s and $\tilde{s}$ are extracted and their balance computed. To avoid redundant patterns, we traverse the pattern tree in a dual way: when visiting a node, we visit also the dual node and flag it as already processed (not output). Note that from an extracted pattern the support of the dual can be retrieved as follows : $|support_{\mathcal{D}}(\tilde{s})| = |support_{\mathcal{D}}(s)| \times \left(\frac{balance(\tilde{s})}{balance(s)}\right)$.

5 Experimental study

We experiment our approach to assess its computational feasibility and, helped by a high level StarCraft II player, the quality of the extracted patterns. These patterns are useful for several tasks among which *on-the-fly win prediction* and *game balance study*. The source code and the data sets are available[4].

Raw replays collection. StarCraft II replays are easily accessible on the Web; several websites offer high level replays in great numbers[5] from which we extracted $371,267$ replays. We are interested in very high level players, since casual (by opposition to professional) players are not able to follow specific strategies. We kept replays involving at least one player in the highest official league (Grandmaster) or playing in average more than 200 *actions per minutes* (APM). At the end, our dataset involves $90,678$ games between $30,678$ players that played an overall total of 3.19 years in game time. The average length of a game is about 20 minutes. Figure 2 (left) gives the average APM during each minute of a game. Figure 2 (right) gives the repartition of the different kinds of actions done in average in time.

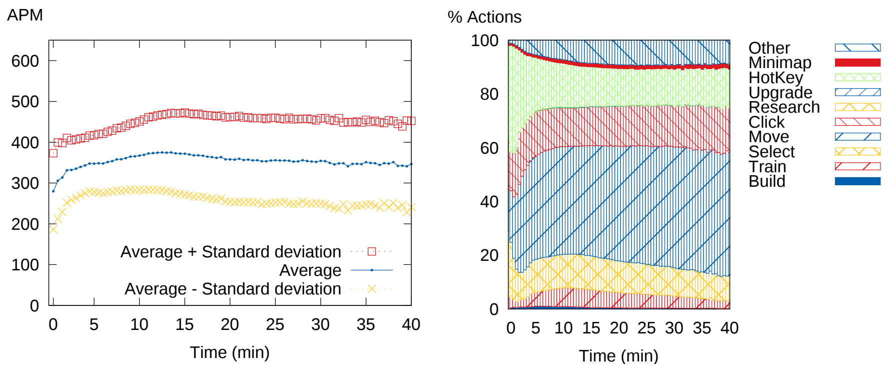


Fig. 2. Different characteristics of the raw replays dataset.

Deriving sequences datasets from raw replays. In StarCraft II, they are different types of games, called match-up, depending on the pair of factions involved, e.g. PvZ describes a game involving a Protoss player against a Zerg player. Each match-up is characterized by different strategies: a Protoss does not play the same strategies when facing either another Protoss player or a Zerg opponent, or even a Terran opponent. As such, we divided the $90,678$ replays into six different sequence datasets, one for every match ups (since there are 3 factions). Buildings are one of the key elements of a strategy [11], since they allow different kinds of units production: From each replay, we derive a sequence where the items represent the buildings the players chose to produce in real time, and itemsets denote windows of time w (See Example 2). We set w to 30

4 http://liris.cnrs.fr/mehdi.kaytoue/sc2
5 http://wiki.teamliquid.net/starcraft2/Replay_Websites

Data	Build				Build + Upgrade + Occurrence			
	Item	Seq.	IS	I/IS	Item	Seq.	IS	I/IS
PvP	1,160	6,668	11.5	2.0	5,624	6,668	11.6	2.1
PvT	3,655	18,754	19.0	2.6	24,019	18,754	19.4	2.7
PvZ	3,748	22,784	19.6	2.7	27,913	22,784	20.1	2.8
TvT	2,201	7,457	20.7	2.8	15,572	7,457	21.2	2.9
TvZ	4,492	23,637	22.5	2.8	33,050	23,637	23.1	2.9
ZvZ	1,689	9,554	14.2	2.2	9,139	9,554	14.7	2.3

Table 1. Sequence database characteristics.

seconds since the ordering of two items over this period of time is insignificant in terms of strategy according to the expert. For the Protoss and Terran factions, we ignored the buildings *Pylon* and *Supply depot* that are insignificant in terms of strategy as well (expert opinion). Table 1 gives the main characteristics of these datasets. We also build a second series of datasets, which includes research upgrades (important elements of strategy) and the number of times the same action appeared, which can better help characterize strategies according to the expert, i.e. $\mathcal{I}$ is defined on $A \times T \times R \times \mathbb{N}$. We wrote a plugin for the Sc2Gears software[6] to parse the binary replay files. This plugin is freely available[7].

Quantitative experiments. We experimented on 2.5GHz and 4GB of RAM machines. We slightly modified the original C++ algorithm PrefixSpan for extracting frequent sequential patterns [6]. Figure 4 (left) shows for the data set PvP that our optimization for computing *balance* speeds up the computation of the balance measure by several orders of magnitude, comparing to the naive approach, making our approach usable for large datasets. It also shows that the time needed to avoid redundancy is negligible, and that the amount of redundant patterns tend to decrease with the minimal support (yet not monotone). Figure 4 (middle) shows that the naive post-processing makes the approach not scalable. In contrast, our optimization allows low minimal supports (Figure 4 (right)). Finally, Figure 3 plots extracted patterns (including redundant) according to their balance measure and support, for the PvP dataset and a minimum support set to 0.1%. Typically, jumping patterns can be observed with a measure equals to 1. Note the "symmetric" aspect of the points cloud, where $y = 0.5$ is the line representing balanced patterns. Note also that empirically, the more frequent the patterns, the closest to 0.5 the balance measure.

Discovering openings. Like in chess game, StarCraft II openings are generally well-known and codified (given a name). We observe that well-known openings in StarCraft II are translated to the most frequent patterns. Starting with the dataset TvZ, we extract frequent sequential patterns with minimum support of 10%. As expected most of the 64 patterns returned are very balanced with values close to 0.5. Openings are *per se* balanced (otherwise the game would not be attractive for the numerous spectators). We filtered the 64 patterns to retain only the 20 that involve actions of both players. We remark for example

[6] https://sites.google.com/site/sc2gears/
[7] http://liris.cnrs.fr/mehdi.kaytoue/sc2/MLSA-PKDD13/Sc2Gears4DM.zip

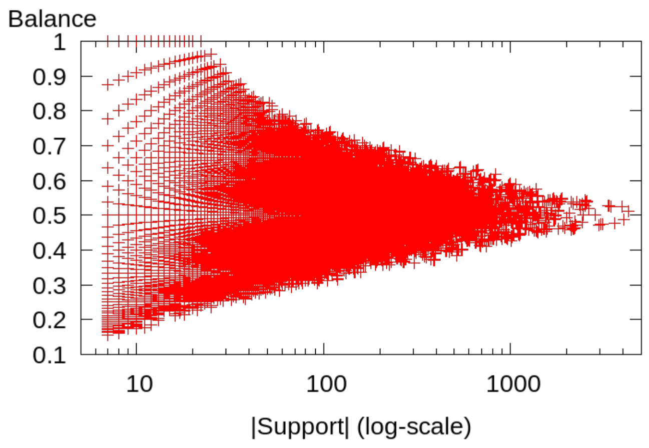


Fig. 3. Balance measure and support of extracted patterns.

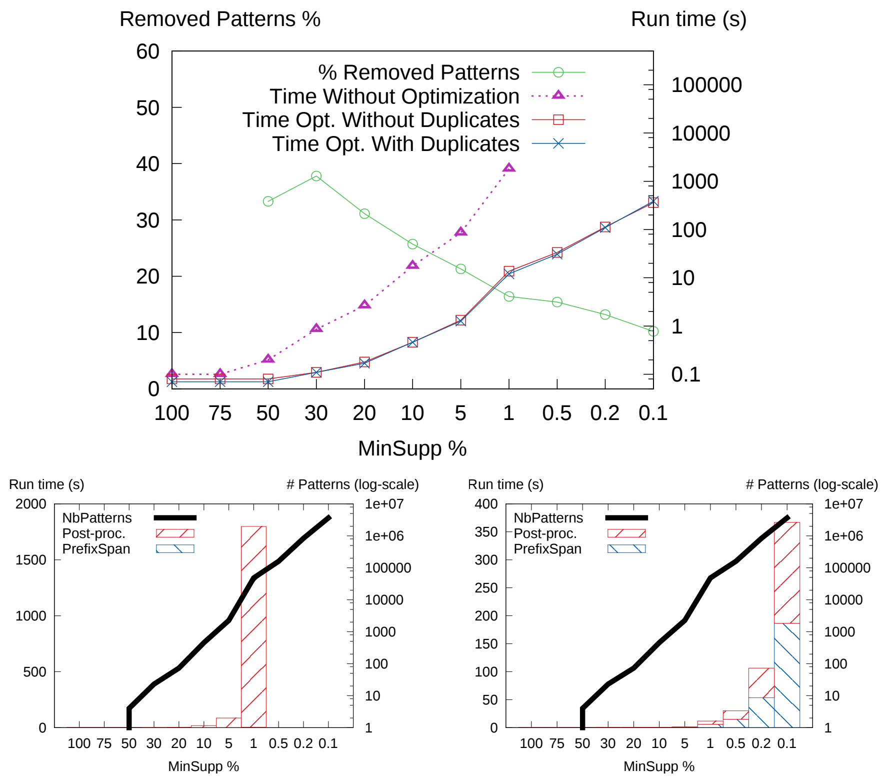


Fig. 4. Execution times and pattern counts

$\langle\{(1, W, Barracks)\}, \{(4, L, Hatchery), (4, L, Sp.Pool)\}\rangle$ (and its dual) which is the well-known opening called *Fast-expand* in the TvZ match-up. Pattern $s = \langle(1, L, Barracks)(4, W, Hatchery)(6, L, ReactorBarracks)(7, L, CCenter)\rangle$ with $balance(s) = 0.5$ denotes also a classical opening of game that leads to a so-called *macro-game*, where players focus on building a strong economy before attacking the opponent. In the PvZ dataset, 49 patterns are extracted with a minimum support set to 10%, and only one pattern involves the two players: the *forge-expand* strategy for the Protoss player (with a balance of 0.506), which is one of the most typical opening in that match-up (supported by 37% of the games).

Discovering unbalanced strategies. Balance is the most important aspect in e-sport: in a match, all opponents should have exactly the same chances to win given the initial conditions. In StarCraft II for example, a faction should not been advantaged compared to another. When a balancing problem is detected, either by the game developers or by the players themselves, the game properties are adjusted to correct this balance issue. We asked the expert for a balance example problem in StarCraft II. The *Bunker-rush* in the TvZ match-ups consists in building, very rapidly, a defensive structure in the opponent's base which causes strong if not fatal damages when not detected in time. We used the second encoding (involving the number of times the bunker has been built until its last appearance) to check that hypothesis and query among the 17,990 patterns ($minSupp = 1\%$). Only 359 patterns involves the *Bunker* building and both players. The top 50 patterns according to *balance* have all a support cardinality higher than 300 and a balance measure higher than 0.61. For example, the 5^{th} pattern clearly indicates that the bunker rush is an imbalanced strategy, even if it clearly appears that the opponent tried to defend against it: $\langle\{(1, W, Barracks, 1)\}, \{(4, L, SpPool, 1)\}, \{(6, W, Bunker, 1), (6, L, SpCrawler, 1)\}\rangle$. This balance issue has been corrected in a patch of the game by changing some properties of the game element.

6 Conclusion

We presented a preliminary work for extracting strategic patterns from replays of the RTS video game "StarCraft II". We introduce a three-steps approach: (i) encode player actions into sequences, (ii) mine sequential patterns, and (iii) compute the balance of each resulting strategy. Through a series of experiments, we show that it is feasible to extract interesting patterns for many e-sport tasks in reasonable time. We believe that the methodology and the results are interesting to help RTS video games editor to balance their games, but also for the players/team managers for analyzing the strategies of a given player over a selected set of replays (e.g. to prepare a competition). Among others, our perspectives of further research include to study the use of other types of discretization techniques for building the sequences, or the use of other types of sequential patterns such as episodes. We will investigate how to achieve *on-the-fly game result prediction*, which consists in using the extracted patterns for predicting the game outcome in real-time. Also, build orders consist in less than 1% of the information contained in a replay. We plan to use other types of actions, and this interestingly comes with problems of pattern mining in noisy data and game state estimation (remembering that the state of the game is never stored in a replay).

References

[1] K. Deng and O. R. Zaïane. An occurrence based approach to mine emerging sequences. In T. B. Pedersen, M. K. Mohania, and A. M. Tjoa, editors, *DaWak*, volume 6263 of *Lecture Notes in Computer Science*, pages 275–284. Springer, 2010.

[2] E. Dereszynski, J. Hostetler, A. Fern, T. Dietterich, T.-T. Hoang, and M. Udarbe. Learning probabilistic behavior models in real-time strategy games. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[3] J. Hostetler, E. W. Dereszynski, T. G. Dietterich, and A. Fern. Inferring strategies from limited reconnaissance in real-time strategy games. *CoRR*, abs/1210.4880, 2012.

[4] J.-L. Hsieh and C.-T. Sun. Building a player strategy model by analyzing replays of real-time strategy games. In *Proceedings of the International Joint Conference on Neural Networks, part of the IEEE World Congress on Computational Intelligence, Hong Kong, China, June 1-6, 2008*, pages 3106–3111. IEEE, 2008.

[5] M. Kaytoue, A. Silva, L. Cerf, W. Meira Jr., and C. Raïssi. Watch me playing, i am a professional: a first study on video game live streaming. pages 1181–1188, 2012.

[6] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In D. Georgakopoulos and A. Buchmann, editors, *ICDE*, pages 215–224. IEEE Computer Society, 2001.

[7] G. Synnaeve and P. Bessiere. A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft. In *Proceedings of 2011 IEEE CIG*, 2011.

[8] G. Synnaeve and P. Bessière. A dataset for starcraft ai & an example of armies clustering. *CoRR*, abs/1211.4552, 2012.

[9] T. L. Taylor. *Raising the Stakes:E-Sports and the Professionalization of Computer Gaming*. MIT Press, 2012.

[10] B. G. Weber and M. Mateas. Case-based reasoning for build order in real-time strategy games. In C. Darken and G. M. Youngblood, editors, *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference, October 14-16, Stanford, California, USA*. The AAAI Press, 2009.

[11] B. G. Weber and M. Mateas. A data mining approach to strategy prediction. In P. L. Lanzi, editor, *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*, pages 140–147. IEEE, 2009.

[12] S. Wender, A. Cordier, and I. Watson. Building a trace-based system for real-time strategy game traces. In *Proceedings of EXPPORT: EXperience reuse: Provenance, Process-ORientation and Traces, ICCBR 2013, Saratoga Springs, NY, USA, 8-11 July 2013.*, 2013.

[13] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large databases. In D. Barbará and C. Kamath, editors, *SDM*. SIAM, 2003.